

FAQ

Q Is SQL a database or a programming language?

A Structured Query Language (SQL) is a programming language. But its focus has been on data since it was invented in the early 1970s to work with IBM's experimental database system. The term SQL is commonly used for databases as well as the language used to manipulate them.

SQL provides facilities for working with data. It is usually bundled with a database application such as the open-source MySQL or Microsoft's SQL Server. These applications modify and extend SQL in different ways, which makes it hard to write standard SQL. However, it is possible to write applications that work with a range of database engines supporting SQL.

Because it is in such wide use, SQL is worth learning and isn't very difficult.

Mike James

IT author, lecturer
and programmer

mike@computershopper.co.uk

Write a custom search engine

Why go to all the bother of programming a search function from scratch? Mike James shows you how to use Microsoft's Desktop Search to make your own search facility



Search engines have revolutionised the way we use the internet and, recently, Windows itself. You no longer need to be careful about where you store your files, because you can find anything simply by searching.

Users expect good search facilities in software, but programming a search function from scratch is a tall order. Luckily, you don't have to, because Microsoft has made the inner workings of its Desktop Search 3.0 available for you to use. You can easily build this into an application that finds and organises documents of all kinds, including photos, music and webpages. With just a little extra effort, you can use it to create a custom search facility for your ASP.NET website.

WHICH DESKTOP SEARCH?

Desktop Search is developing rapidly, but the one to use at the time of writing is version 3.01. This is built into Vista, but if you are using Windows XP or Windows 2003 then you will have to download and install it. A good place to visit to find out about Desktop Search and the relevant downloads is www.microsoft.com/windows/desktopsearch. There is also an SDK, which includes some examples and a .NET wrapper for most of the COM objects involved in the API.

Unfortunately, most of the interfaces have no documentation and this makes the wrapper difficult to use. If you simply want to use the Desktop Search to find files, you probably don't need the wrapper, as this portion of the API provides a simulated database interface. This means you can use a SQL query to search for documents based on a wide range of

properties and content. Before we get started, it is worth understanding how the search works.

After you've installed Desktop Search, it starts to build an index of files. It doesn't index the entire disk, however. The default settings are to index your document directory and the directory in which Outlook stores emails. If you store files in a directory outside your documents directory – on another disk, say – then these will not be included in the index. You can configure the search engine to include these directories, but it is important to remember that just because the Desktop Search cannot find a file, this doesn't mean it isn't on the disk.

To build the index, the search engine has to examine every file. It does this with the help of IFilters, which return properties such as the filename, owner, creation date and, in some cases, the text the file contains. The index takes time to create, and until it has been completed you cannot trust the results of a search. Once the index has been created, the search engine keeps it up to date by examining any new files or changed files. This runs as a low-priority task and indexing is suspended while you are using the machine; this means the very latest files you have created or downloaded may not be found in the index.

UNDERSTANDING THE DATASET

The index created by Desktop Search can be queried as if it were an OLE DB database. Microsoft's newest database programming framework, ADO .NET, is perfectly happy working with OLE DB, so it seems sensible to use this up-to-date approach in our project.

If you don't know how ADO .NET works, using it with Desktop Search is a good introduction.

Start a new Windows Application C# project called SSearch using C# Express Edition or a full copy of Visual Studio. Add this line to the start of the code:

```
using System.Data.OleDb;
```

You need to place a button on the form to run our search code.

You can work in ADO .NET using typed or untyped database objects. We will use typed objects, which require more effort to create, but save time later on and also make it easier for Visual Studio to spot bugs in your code.

The main database object of ADO .NET is the DataSet, which can contain a number of DataTables. Each DataTable object roughly corresponds to a database table with rows and columns. In a complex situation, the tables in a DataSet can be linked to form a relational database, but it is stored in memory rather than on disk. For our project, we need only a single table: the result of a query on the Desktop Search index.

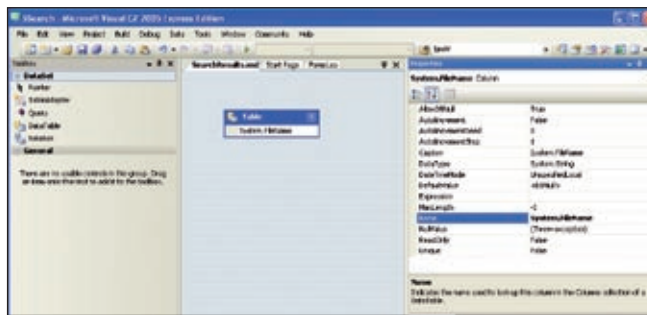
Our first job is to create a suitable DataSet to hold the results of the query. As mentioned, there are two options: typed or untyped. The typed approach must define the columns that the search query will return, but a wizard helps us with this. The wizard generates a class that inherits from DataSet adding various typed properties – one for each table and each column we define. Let's see how easy this is in practice.

Use the command Project, Add New Item and select DataSet in the dialog box that appears. Call the new DataSet SearchResults.xsd. This creates an XML file that will be used to store the database definition, or schema. The XML file is used to generate the new class.

After you've added the new DataSet you will see the DataSet Designer, which you use to add tables and then columns to the tables. We need only a single table called Table. It has to be called Table because this is the name returned by the index query. Add this table by right-clicking or dragging a table from the toolbar. Once you have changed the table's name, you can add a column called System.FileName, which is specified as a string by default. Later, you can add additional columns to the table, but for the moment we are only going to use the FileName column of the query result, so this is all we need.

MAKING THE CONNECTION

The typed SearchResults DataSet is an in-memory version of our database, but to get



◀ You can use the Designer to add a table and a single column

some data into it we need to make a connection to the real database. This is achieved using a connection object. As we are working with an OLE DB data provider, we need an OleDbConnection object; there is a specific connection object for each of the traditional data providers. In most cases, it is possible to use the same wizard that created the DataSet to create the connection object. However, in this case, because the database doesn't exist as a file on disk, we have to do the job manually. Fortunately this isn't difficult.

The connection is specified by a connection string, which you usually have to look up. In this case, all we need is:

```
string connectionString =
    "Provider=Search.CollatorDSO;
    Extended Properties='Application
    =Windows'";
```

Now we can create the connection object and store the connection string in it:

```
OleDbConnection SearchConnect = new
    OleDbConnection();
SearchConnect.ConnectionString =
    connectionString;
```

TRANSFERRING DATA

As well as a connection object to connect to the database you need a DataAdapter, which is responsible for transferring data to and from the database. The DataAdapter is the place where you store the SQL query that extracts a subset of the data into a suitable DataSet object. As we are working with an OLE DB data provider, we need an OleDbDataAdapter. As in the case of the connection object, there is a special adaptor for each type of data provider:

```
OleDbDataAdapter SearchAdpt = new
    OleDbDataAdapter("SELECT Top 5
    System.FileName FROM SYSTEMINDEX",
    SearchConnect);
```

The first parameter of this constructor is the SQL query string that specifies the records we want to return. If you know SQL, you will recognise the SELECT statement. Top 5 just returns the first five records, and only the System.FileName column is returned from the table called SYSTEMINDEX. Later, we can create some more interesting queries, but this is sufficient to show that it all works.

Everything is ready to retrieve the data, but we still need somewhere to store it. An instance of the SearchResults dataset is easy to create:

```
SearchResults SearchData = new
    SearchResults();
```

To fill it with data that matches the query specified in the DataAdapter, we will use the Fill method:

```
SearchAdpt.Fill(SearchData);
```

There are Update and Delete methods of the DataAdapter that modify the database using the contents of the DataSet, but the search index is read-only so they don't work.

If you run the program now and click the button, the query will be run on the index and the names of the first five folders/files will be transferred to the DataSet. To access data in the DataSet, you use its properties. For instance:

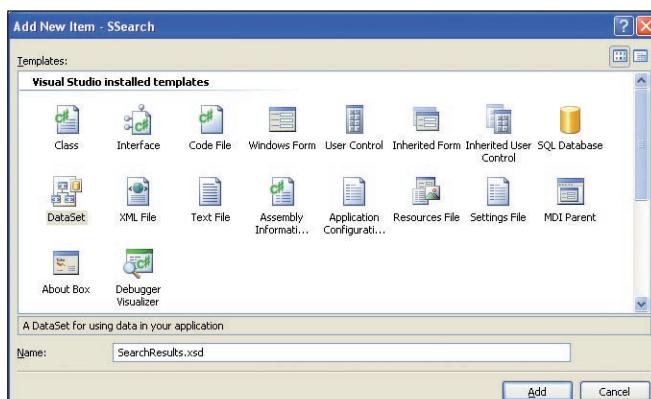
```
string temp = SearchData.Table[0].
    _System.FileName;
```

Here the SearchData.Table part selects a table called Table in the DataSet SearchData. If there were several tables in the DataSet, they would each be accessible as a property of SearchData with the same name. Next, the index [0] selects the first row of the table. The rows are numbered starting from zero. Finally, the _System.FileName part selects the column corresponding to System.FileName. The only complication is that the search index uses full stops in the names of its fields and ADO can't use these within the names of properties. To solve the problem, all full stops are converted to underscores.

If you put a break point in the program just after the above line, you can stop the program and examine the contents of temp. However, we are now going to look at a more sophisticated way of examining the results.

BINDING TO A GRID

ADO .NET provides everything you need to work with databases without having to start from scratch. For example, now that we have a DataSet, we may want to transfer all or some of the data to some sort of tabular control so the user can view and perhaps modify it. Transferring data to and from such a control isn't difficult,



◀ Add a new typed DataSet to the project

MICROSOFT PRESS

Microsoft Windows PowerShell Step by Step

★★★★

£20

What should we make of Windows PowerShell, Microsoft's new scripting language for system administrators? This feature has been missing from Windows for such a long time that it may be too late to build a significant following. PowerShell uses object-oriented techniques to bring the kind of automation first seen in DOS batch files into the 21st century. You can download and install it for free from the Microsoft website, but why bother?

Ed Wilson attempts to fire our enthusiasm by providing lots of useful examples in this new book. He covers projects from the usual system checking and repetitive tasks to more advanced topics such as WMI, Active directory, ADO and Exchange 2007. Of these, the only area that makes a good argument for PowerShell is Exchange 2007 – mainly because the Exchange design team have decided to use it.

SUMMARY

- ✓ Plenty of useful examples
- ✗ Can be hard to follow

SUPPLIER www.amazon.co.uk
ISBN 0735623953
PAGES 320



The book's task-oriented, step-by-step approach isn't the natural way to learn a new language. If you don't program already, you will find it tough going, as it introduces PowerShell's programming concepts in a random order. The big problem is that PowerShell doesn't seem to offer anything that others such as VBScript and JScript don't already have. As if to make the point, an appendix covers how to translate VBScript into PowerShell code, but this bit is already available on the Microsoft website. The CD bound into the back provides all the samples and many additional utilities. This is certainly a useful book, but we are not so sure about the technology.

but it is very tedious. Happily, ADO .NET provides a DataGridView control that does away with all the tedious programming by automating the transfer.

If you open the Form Designer and place a DataGridView on the Form, you can bind it to a data source. If you click on the control's Task List, the small arrow icon in the top right-hand corner, you can select a data source from within Project Data Sources and, in this case, select Table within SearchResults. This causes the DataGridView to display a single column and a dummy row corresponding to the definition of Table within SearchResults.

MAKING THE BINDING WORK

Two more things happen when you set the data source. A DataSet control called searchResults is added to the form, and a BindingSource control is also added. You needn't worry about what these do, as they are just part of the automatic plumbing that transfers the data. The only minor

complication is that you have to work with the automatically generated DataSet object (searchResults) in your program, rather than creating your own DataSet manually.

To make the binding work, first remove:

```
SearchResults SearchData = new
SearchResults();
```

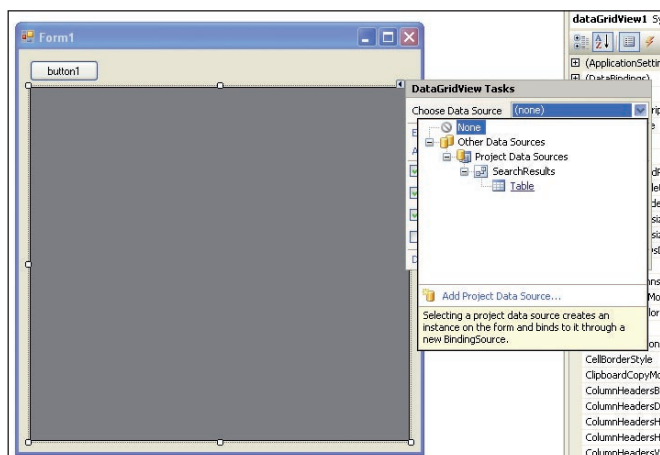
Then edit any references to SearchData in the program. Change:

```
SearchAdpt.Fill(SearchData);
```

This needs to read:

```
SearchAdpt.Fill(searchResults);
```

Now when you run the program and click the button, you will see the grid populate with the results of the search. This demonstrates just how easy object-oriented programming can be.



◀ Binding a DataGridView to a data source is as easy as picking from a list

MORE PROPERTIES

Now we have a basic search working, doing more is just a matter of adding columns to the typed DataSet using the wizard and building appropriate SQL queries. This means learning more about SQL's SELECT command. The general format of this is:

```
SELECT [TOP <positive integer>] <columns>
FROM [machinename.]SystemIndex
[WHERE <conditions>]
[ORDER BY <column>]
```

You can leave out any of the clauses surrounded by square brackets and still have a valid SELECT. There are a few additions to standard SQL that you can use with Desktop Search. Equally, there are some things from standard SQL that don't work for Desktop Search. You can use the usual wildcard characters; for example, * matches any number of characters and ? matches any single character. You cannot, however, use the standard SQL shorthand of * to specify 'return all columns'. You have to explicitly list all the columns you want. The WHERE clause can be used to specify what you are looking for, and you can build logical expressions using AND, OR and NOT.

You can search on Properties, which are columns in the database, or on the full text of an indexed file. For example, to find all files ending in '.wma' you would use:

```
SELECT System.FileName FROM
SYSTEMINDEX WHERE System.
FileExtension = '.wma'";
```

To perform the same search, but return additional columns, you would use something like this:

```
"SELECT System.FileName, System.
ItemPathDisplay, System.DateCreated
FROM SYSTEMINDEX WHERE System.
FileExtension = '.wma'";
```

The biggest problem is discovering what properties are available. In principle, you should be able to work with all the shell properties, which you can find listed at Microsoft's website (<http://tinyurl.com/2m9az9>). In practice, however, not all of them work and you have to use trial and error to see which do.

UPDATING THE DATAADAPTER

Once you've formulated the SQL query you need to return the search results you want, you need to update the definition of the DataAdapter. For example, to return three columns, two strings (FileName and ItemPathDisplay) and a Date (DateCreated), you need:

```
OleDbDataAdapter SearchAdpt = new
OleDbDataAdapter("SELECT System.
FileName, System.ItemPathDisplay,
System.DateCreated FROM SYSTEMINDEX
WHERE System.FileExtension = '.wma'",
SearchConnect);
```

For this to work, we need to modify the definition of the strongly typed DataSet. All you have to do is use the DataSet Designer to add the two new columns, making sure you enter the exact names. In addition, you also have to set the DataType property of System.DateCreated

to `System.DateTime`; you can pick this from the drop-down list in the Properties window. In general, you have to set the `DataType` and perhaps additional properties for any new columns you add unless the defaults happen to work. When you leave the Designer, the definition of the `DataSet` will be updated to include the new columns as appropriate new typed properties.

In many cases, this is all you have to do. However, if you have controls bound to the typed `DataSet` then you need to refresh this binding. Go to the Form Designer and delete the `DataSet` and the `BindingSource` controls – named `searchResults` and `tableBindingSource`, in this case – and then rebind the grid to `Table` in the `DataSet`. It's a small inconvenience, considering the automatic generation of the code you get to transfer data from the `DataSet` to the grid.

When you next run the program, you will see the grid has three columns, and you should find data in each of them.

NARROWING YOUR SEARCH

If you want to look in a particular directory for a file, you could include a `WHERE` condition on the `System.ItemPathDisplay` property. This would search everywhere, but only return documents that had a path that matched the specification. However, there is a more efficient way.

The `SCOPE` or the `DIRECTORY` keywords can be used in a Desktop Search SQL query to control where the search is performed. The difference between the two is that `SCOPE` does a deep search, including all subfolders, whereas `DIRECTORY` searches just the folder you specify. For example, say you add this to your query:

```
"WHERE SCOPE='file:C:\Files\Reports'"
```

This searches `C:\Files\Reports` and all the subfolders it contains. However, say you add:

```
"WHERE DIRECTORY='file:C:\Files\Reports'"
```

This searches just `C:\Files\Reports`. `SCOPE` and `DIRECTORY` don't work as described in the Microsoft documentation. There is no need to put `//` after `file`: and you can use `/` or `\` in the path specification.

However, when using `C#`, you need to stop the language from interpreting the `\` characters as escape codes. To do this, you must add `@` to the start of the whole string so, for instance, the full query reads:

```
@ "SELECT System.FileName, System.ItemPathDisplay, System.DateCreated
FROM SYSTEMINDEX WHERE SCOPE='file:
C:\Files\Reports'"
```

MANAGING CONTENT

The real power of the Desktop Search engine is that it indexes the contents of files. If you want to search the contents of indexed files, you need to know about the `FREETEXT` and `CONTAINS` keywords. `FREETEXT` is the simpler of the two and it enables you to search for a fixed phrase. For example, consider:

```
WHERE FREETEXT('Computer Shopper')
```

This searches for any file that contains 'Computer' or 'Shopper', or both. You can also add a column specification so that properties as well as content are searched for these words. For example, to search all the properties and content you would use:

```
WHERE FREETEXT(All,'Computer Shopper')
```

`FREETEXT` uses a variety of matching algorithms and returns a measure of how well the document matches the search criteria in `System.Search.Rank`, which you can retrieve and use. You can modify the way the rank is calculated using the `RANK BY` clause; see the documentation for more information.

The `CONTAINS` keyword is more suitable when looking for exact matches. You can specify what you are looking for using `AND`, `OR` and `NOT`, and use wildcards. For example, to find any word starting with `Shop`, you would use:

```
WHERE CONTAINS('Shop*')
```

You can use `NEAR` to find occurrences of one word near another. Use `FORMSOF` to match for words and inflections of the word. So to find documents containing `run`, `ran` or `running`, use:

```
CONTAINS('FORMSOF(INFLECTIONAL,run)')
```

Full text searches are clearly powerful, but a programmed search can generally do more by including conditions on properties such as `System.Author` or `System.DateCreated`.

SEARCHING TIMES

The Desktop Search facility is powerful and we have only scratched the surface here. However, we have looked at most aspects of its operation and proved that databases are easy. You should be able to follow the documentation on the Microsoft website and create custom search applications that do exactly what you need. 

CROWN PUBLISHERS Dreaming in Code

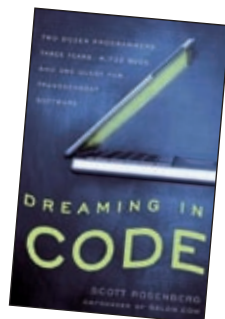
★★★★

£12

Ever wondered what it's like to develop open-source software? Scott Rosenberg has written a journal of the Chandler project, subtitled *Two Dozen Programmers, Three Years, 4,732 Bugs And One Quest For Transcendent Software*. This book documents the personalities and the places where the work was done in an entertaining way. However, it's also a frustrating read, as our heroes struggle with abstract concepts and fail to hit self-imposed deadlines.

The Chandler project is idealistic, with the focus on creating great code. This means it's free from the pressures of management or commercial goals. The real surprise is that this seems to make the task harder, not easier. Along the way, many of the pitfalls of software development are explored via accounts of projects that failed, attempts to make the programming easier and plenty of historical context. You will also learn about the difficulties of measuring progress, tracking and dealing with bugs, the philosophy of open source and how personality is an important factor in building great code.

This project is a world away from how most programmers work, but it is interesting enough to recommend to the non-programmer who wonders why programming is so hard.



SUMMARY

-  An insight into a real-world project
-  A frustrating read

SUPPLIER www.amazon.co.uk
ISBN 1400082463
PAGES 416

ADDISON-WESLEY VBScript, WMI, and ADSI Unleashed

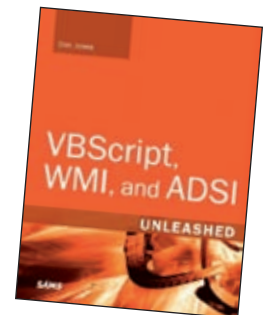
★★★★★

£25

VBScript is the only product Microsoft has officially finished. No further versions are planned, but it will continue to ship in all versions of Windows. Its main use is to create administrative scripts that automate routine tasks. Microsoft recently introduced PowerShell as an alternative (see review on page 162), but there are lots of VBScripts out there and it's still worth learning all about it.

Don Jones does a good job of introducing the basics of the language with lots of examples of what it can do for you. The first 150 pages introduce variables, flow of control and what you need to know in any programming language. After that, the book is VBScript and Windows specific. Once we have looked at the range of scripting objects available, we look at WMI and ADSI, core Windows administration facilities. As the book progresses, examples are used to expand the reader's knowledge of VBScript and programming in general – writing subroutines, testing, debugging, efficiency and so on. The explanations are clear and the presentation logical.

If you are a beginner it's an ideal approach; if not, this book still has something to offer.



SUMMARY

-  Clear, with helpful examples
-  Slow-paced if you're not a beginner

SUPPLIER www.amazon.co.uk
ISBN 0321501713
PAGES 555